The First Report of PRML

Liu Jitao
22371262@buaa.edu.cn

In this report, I am going to use 3 liner models and 3 non-linear models to fit a set of data consisting of 100 elements. Each part involves a short introduction of the models and the results are shown in figures. All the codes are published on `https://github.com/KevinTJL/PRML-2025/tree/main/work_1`.

I. Results to Question1

A. LSM

LSM is widely used to fit one-dimensional functions. Given a datasets with n observations (x_i , y_i), where y_i are the observed values and $f(x_i; \theta)$ is the model's predicted value parameterized by θ , the least squares objective function is:

$$S(\theta) = \sum_{i=1}^{n} (y_i - f(x_i; \theta))^2 \tag{1}$$

We can minimise the $S(\theta)$ to fit the function.

By using numpy library function we can easily build LSM model. Here is my result by using it to fit the training data and the testing data.

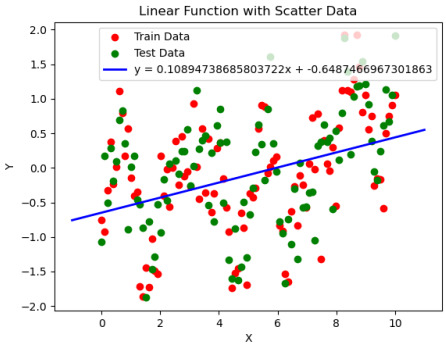

Fig. 1 This picture shows the fitness of LSM method with training data and test data

It is clear that the LSM method cannot fit non-linear data appropriately. Also the fit function is :

$$y = 0.1089 * x - 0.6487 \tag{2}$$

B. Gradient Descent

Gradient Descent(GD) is another popular method in function fitting. It is an iterative optimization algorithm used to minimize a differentiable function $f(\theta)$ and works by updating parameters in the direction opposite to the gradient (i.e., the steepest ascent) to find the function's minimum:

$$\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t) \tag{3}$$

θ_t is the parameter at iteration t , η is the learning rate (step size), $\nabla f(\theta_t)$ is the gradient of the function at θ_t . In this method, I use the Euclidean norm as the loss function which can accurately reflect the error. The learning rate and the number of iterations are, respectively, 0.01 and 1000. The results are shown below. The polynomial to the highest power is 1.

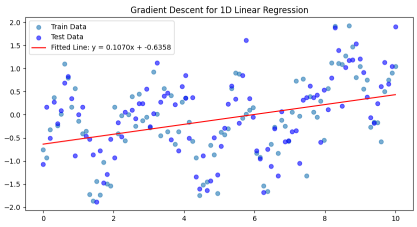

Fig. 2 This picture shows the fitness of GD method with training data and test data

The fitted equation is :

$$y = 0.1070 * x + -0.6358 \tag{4}$$

It is clear that the GD method cannot fit non-linear data appropriately as well. Besides, if the polynomial to the highest power is greater than 1, there will not be any result due to gradient explosion.

C. Newton method

Newton method is much more often to fit high-dimension functions. Its performance in fitting one-dimension function is not good enough. Here is the result.

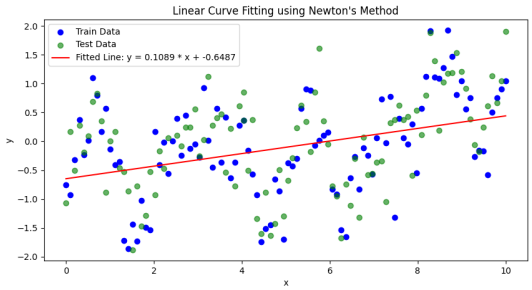

Fig. 3 This picture shows the fitness of Newton method with training data and test data

The fitted equation is :

$$y = 0.1089 * x + -0.6487 \tag{5}$$

And the result is not good.

II. Results to Question2

As the previous results show, it is almost impossible to fit the given data with liner models. And I am going to use 2 non-liner models to solve the problem.

A. Random Forest Regression

Random Forest Regression (RFR) is an ensemble learning method based on Decision Trees, which uses multiple decision trees for regression prediction and takes the average of multiple trees to improve the stability and generalization ability of the model. By using the RandomForestRegressor function from sklearn library function, we can easily realize it.

The curve reflects the characteristics of the data, but it is not smooth.

Fig. 4 This picture shows the fitness of RFR method with training data and test data

B. Adaptive polynomial regression

Polynomial regression makes curves smoother and easier to implement. However, it is not easy to determine the power of polynomials, so the cross-validation method is introduced, which can realize the adaptive determination of the number of arbitrary input data, so as to improve the efficiency.

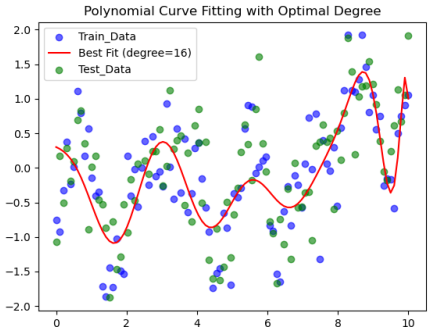

Fig. 5 This picture shows the fitness of APR method with training data and test data

The equation is :

$$\begin{aligned} y =& 0.3008 - 0.2194 * x^1 - 0.3190 * x^2 - 0.3264 * x^3 - 0.1959 * x^4 + 0.0674 * x^5 \\ &+ 0.2592 * x^6 + 0.0581 * x^7 - 0.2752 * x^8 + 0.1849 * x^9 + -0.0641 * x^{10} \\ &+ 0.0137 * x^{11} - 0.0019 * x^{12} + 0.0002 * x^{13} + -0.0000 * x^{14} + 0.0000 * x^{15} + -0.0000 * x^{16} \end{aligned} \tag{6}$$

C. Support Vector Regression

Support Vector machine Regression (SVR) is a regression algorithm based on support vector machines (SVMS) that is used to fit complex non-linear data while controlling the complexity of the model and reducing overfitting. It aims to solve:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{7}$$

subject to:

$$\begin{aligned} y_i - w^T\phi(x_i) - b &\le \epsilon + \xi_i \\ w^T\phi(x_i) + b - y_i &\le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* &\ge 0 \end{aligned} \tag{8}$$

ϵ is the margin within which predictions are error-free. ξ_i, ξ_i^* are slack variables for handling outliers. C is a regularization parameter controlling bias-variance tradeoff.

sklearn library function provide a method to achieve it. I use RBF kernel to deploy the model. Here is the figure.

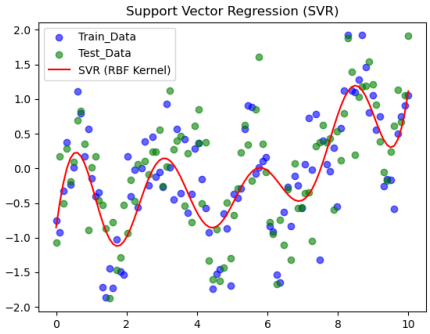

Fig. 6 This picture shows the fitness of SVR method with training data and test data

III. Evaluations and Conclusion

To compare the performance of all models, we introduce mean square error, root mean square error and mean absolute error in the evaluation part. This table shows the performance of all 6 models under different measurement:

	Train Error			Test Error		
Method	MSE	RMSE	MAE	MSE	RMSE	MAE
LSM	0.6134	0.7832	0.6339	0.595	0.7714	0.6418
GD	0.6134	0.7832	0.6339	0.595	0.7714	0.6418
Newton	0.6134	0.7832	0.6339	0.595	0.7714	0.6418
RFR	0.1282	0.358	0.2791	0.2748	0.5242	0.423
APR	0.3309	0.5752	0.461	0.3607	0.6006	0.4669
SVR	0.3292	0.5738	0.4474	0.3399	0.583	0.4625

Table 1 Model Performance Comparison

This experiment compares the performance of six regression models: LSM, GD, Newton, RFR, APR, and SVR. The evaluation metrics include mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) on both training and test datasets. The results indicate that RFR achieves the lowest error values on the training set, suggesting a strong fitting capability. However, SVR and APR exhibit more balanced performance between training and test errors, demonstrating their robustness and generalization ability. Given their smooth fit and relatively lower test errors, SVR and APR are recommended as the most effective models for this problem.